Available online at www.sciencedirect.com

SciVerse ScienceDirect

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 276 (2011) 81–104

www.elsevier.com/locate/entcs

Towards Effects in Mathematical Operational Semantics

Faris Abou-Saleh Dirk Pattinson

Department of Computing
Imperial College London

Abstract

In this paper, we study extensions of mathematical operational semantics with algebraic effects. Our starting point is an effect-free coalgebraic operational semantics, given by a natural transformation of syntax over behaviour. The operational semantics of the extended language arises by distributing program syntax over effects, again inducing a coalgebraic operational semantics, but this time in the Kleisli category for the monad derived from the algebraic effects. The final coalgebra in this Kleisli category then serves as the denotational model. For it to exist, we ensure that the the Kleisli category is enriched over CPOs by considering the monad of possibly infinite terms, extended with a bottom element.
Unlike the effectless setting, not all operational specifications give rise to adequate and compositional semantics. We give a proof of adequacy and compositionality provided the specifications can be described by evaluation-in-context. We illustrate our techniques with a simple extension of (stateless) while programs with global store, i.e. variable lookup.

Keywords: Operational semantics, Coalgebras, Effects, Kleisli category, Comodels

1 Introduction

Operational and denotational semantics provide orthogonal descriptions of a programming language: operational semantics specifies the execution of programs, and denotational semantics associates a mathematical meaning to programs. Crucially, denotational semantics is *compositional* so that the denotation of a program can be inferred from the interpretation of its subprograms, and so facilitates the task of compositional verification, either of program properties, or of features of the language at large. The operational and denotational description of a language are connected via the notion of adequacy – operationally equivalent programs should have the same denotation, and vice-versa. The verification of adequacy is usually done on a language-by-language basis and is often tedious for non-trivial languages.

A more general approach was proposed by Turi and Plotkin [24] in the case of structural operational semantics, assuming that the operational behaviour of a language can be modelled coalgebraically. Then, the semantic domain can be taken as a final coalgebra, and mapping programs to their denotations provides an

doi:10.1016/j.entcs.2011.09.016

adequate denotational semantics. The main result of *op.cit.* is that this semantics is compositional if the operational rules can be given in the form of a natural transformation of syntax over behaviour, i.e. they form an *abstract operational semantics*.

While this programme of mathematical operational semantics is well-suited to process-calculi-like languages, it is not yet clear how imperative languages, or indeed any language that produces computational effects [14,15,5], can be treated adequately, and a number of difficulties present themselves. Firstly, giving an abstract operational semantics relies on syntax and behaviour being defined on the same category, whereas computational effects are are most naturally expressed in the Kleisli category of a monad [12], which is unsuitable for representating syntax. Second, many of the standard techniques for constructing final coalgebras (that play the roles of denotational domains) fail in a Kleisli category. Finally, working with a Kleisli category to obtain adequate denotational models, we need new proof techniques to show compositionality of the denotational semantics.

In this paper, we make first steps towards abstract operational semantics for languages with effects and address the difficulties outlined above. We begin with a pure, effectless language, and an operational semantics given by a natural transformation. We extend the syntax with computational effects as given by an algebraic theory [19], and extend the operational semantics to handle these effects; this induces an effectful operational model by structural recursion.

To obtain the extended operational semantics, we introduce a notion of 'dependency-support' that records which subterms appear in the premises of the operational rules. In particular, this covers the case where operational rules have at most one premise that exhibits nontrivial behaviour.

Once it is obtained, an effectful operational model is considered as a coalgebra in the Kleisli category induced by the algebraic effects. This ensures that when the coalgebra behaviour is iterated, one accumulates the effects arising during program execution. The existence of final coalgebras (denotational models) in this setting is subject to this Kleisli-category being enriched over chain-complete partial orders; we explore the issue of satisfying this requirement in a generic way.

Lastly, unlike the effectless setting, we demonstrate that not all abstract operational semantics are compositional. We give some abstract conditions that guarantee compositionality; in particular, we show that operational semantics described by evaluation-in-context give rise to compositional semantics.

At this point, our approach has several limitations. We neglect equational specifications of algebraic effects which results in a too fine-grained notion of operational equivalence. Moreover, one often wishes to abstract away some aspects of program execution traces, such as the number of steps before termination; we do not do this here. We plan to address these points in future work following [18] along with more discussion of comodels [16].

Related Work. Algebraic effects have been considered in [14] in the context of PCF, which has recently been extended to account for a larger class of operational phenomena [8]. Both papers are mainly operational, and indeed [14] concludes

with the remark that 'one would wish to reconcile this work with the co-algebraic treatment of operational semantics' in [24]. The framework of abstract operational semantics itself has been extended into various directions [9,10], but effects have so far been elusive. Our treatment of computational effects is inspired by [19,6]. Final coalgebras in Kleisli categories were studied in [4].

2 Introducing Effects into Syntax and Behaviour

We recall the mathematical operational semantics of Turi and Plotkin [23,24] applied to multi-sorted syntax signatures in a category $\mathcal{C}^{\mathbb{S}}$. In general, we assume that $\mathcal{C}$ is cartesian closed, with countable products and coproducts (ω-arities are needed for effects, e.g. the read-operation of [15]). We also assume that countable polynomial functors have initial algebras and final coalgebras.

Definition 2.1 *If $F : \mathcal{C} \to \mathcal{C}$ is a functor, an* F-algebra *is a pair $\langle A, \alpha\rangle$ where $A \in \mathcal{C}$ and $\alpha : FA \to A$. Dually, an* F-coalgebra *is a pair $\langle C, \gamma\rangle$ where $\gamma : C \to FC$. Algebras and coalgebras form categories that we denote by $\mathsf{Alg}(F)$ and $\mathsf{Coalg}(F)$, respectively. An* $\mathbb{S}$-sorted signature Sig *over a set $\mathbb{S}$ of sorts has function symbols with (at most countable) arities, written as $f : (s_i)_{0\leq i<\alpha} \to s_f$ where $s_i \in \mathbb{S}$, and $\alpha \leq \omega$ is the arity of f. If arities are clear from the context, we write $f : (s_i) \to s_f$. Every $\mathbb{S}$-sorted signature Sig induces a functor $\Sigma_{\mathsf{Sig}} : \mathcal{C}^{\mathbb{S}} \to \mathcal{C}^{\mathbb{S}}$ given by $\Sigma_{\mathsf{Sig}}(X_s)_{s\in\mathbb{S}} = (\coprod_{f:(s_i)\to s} \prod_{i<\alpha} X_{s_i})_{s\in\mathbb{S}}$ where α is the arity of f.*

For our running example, we take while programs in $\mathcal{C} = \mathsf{Set}$ that we describe via an extension of an effectless language – the fragment of while without variable lookup x or update $x{:=}n$ – with effects for global state. The base language that we call *stateless while* mainly serves the purpose of exemplifying our techniques.

Example 2.2 Let $\mathbb{S} = \{\mathbf{Num}, \mathbf{Bool}, \mathbf{Prog}\}$ denote numerical, boolean and program expressions, respectively. The signature of *stateless while* is given by

$$
\begin{aligned}
N &::= n \mid N + N \mid N * N \mid +_n(N) \mid *_n(N)\\
E &::= b \mid N = N \mid N \leq N \mid =_n(N) \mid \leq_n(N) \mid \neg E \mid E \wedge E\\
P &::= \mathtt{skip} \mid P\,;P \mid \mathtt{while}\ (E)\ \mathtt{do}\ \{P\} \mid \mathtt{if}\ (E)\ \mathtt{then}\ \{P\}\ \mathtt{else}\ \{P\}
\end{aligned}
$$

where n is a numeral in $\mathbb{N}$, and b is a boolean in $\mathbb{B} = \{true, false\}$. The auxiliary operators $+_n(-)$, $*_n(-)$, etc. record partial results of evaluating expressions like $N + M$ from left-to-rightAs explained in Section 3, these auxiliary operators are introduced to give a correct effectful operational semantics for $+$ and $*$. The grammar above induces a syntax functor $\Sigma : \mathsf{Set}^3 \to \mathsf{Set}^3$ in a straightforward way.

Remark 2.3 *If Σ is a syntax functor induced by a signature, the assumptions on $\mathcal{C}$ provide us with a free construction $F \dashv U$ where $U : \mathsf{Alg}(\Sigma) \to \mathcal{C}^{\mathbb{S}}$ is the forgetful functor on Σ-algebras. We write $T_\Sigma = UF$; intuitively, the s-component $(T_\Sigma X)_s$ contains the terms of sort s over sorted variables (X_s).*

Operational models of languages given by a signature Sig are coalgebras $T_\Sigma 0 \to BT_\Sigma 0$ where $B : \mathcal{C}^{\mathbb{S}} \to \mathcal{C}^{\mathbb{S}}$ is an endofunctor describing the *behaviour* of programs and $T_\Sigma 0$ is the set of closed programs. Their operational semantics may be specified by an abstract operational semantics (abstract OS):

Definition 2.4 *An abstract operational semantics (abstract OS) for syntax and behaviour functors Σ, B is a natural transformation $\rho_X : \Sigma(X \times BX) \longrightarrow BT_\Sigma X$.*

Example 2.5 The observable behaviours of *stateless while* are deterministic transitions that may produce a value (for boolean and arithmetic expressions) whereas for programs, only termination is observable. That is, we let $B(N, E, P) = (N + \mathbb{N}, E + \mathbb{B}, P + 1)$, or equivalently $(N, E, P) + (\mathbb{N}, \mathbb{B}, 1)$.

For numeric expressions $u, u' \in N$, we write $u \to u'$ to represent a transition from u to u' – formally, u has behaviour $\mathsf{inr}(u') \in N + \mathbb{N}$. We write $u \xrightarrow{\checkmark} \underline{n}$ if u terminates with value $n \in \mathbb{N}$ – formally $\mathsf{inr}(n) \in N + \mathbb{N}$. Similar notation applies for other types. We write $p \xrightarrow{\checkmark}$ if program $p \in P$ terminates.

The abstract operational semantics for stateless while may be presented by standard operational rules such as:

$$\frac{}{n \xrightarrow{\checkmark} \underline{n}} \quad \frac{}{b \xrightarrow{\checkmark} \underline{b}} \quad \frac{}{\texttt{skip} \xrightarrow{\checkmark}} \quad \frac{u \to u'}{u + v \to u' + v} \quad \frac{u \xrightarrow{\checkmark} \underline{n}}{u + v \to +_n(v)} \quad \frac{v \to v'}{+_n(v) \to +_n(v')} \quad \frac{v \xrightarrow{\checkmark} \underline{m}}{+_n(v) \xrightarrow{\checkmark} \underline{n+m}}$$

$$\frac{e \to e'}{\texttt{if}\ (e)\ \texttt{then}\ \{p\}\ \texttt{else}\ \{q\} \to \texttt{if}\ (e')\ \texttt{then}\ \{p\}\ \texttt{else}\ \{q\}} \quad \frac{e \xrightarrow{\checkmark} \underline{true}}{\texttt{if}\ (e)\ \texttt{then}\ \{p\}\ \texttt{else}\ \{q\} \to p} \quad \text{(etc.)}$$

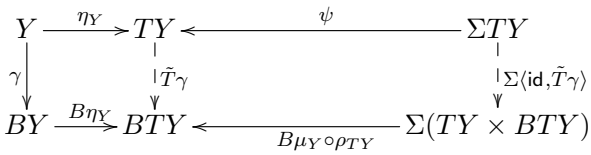
$$\frac{p \to p'}{p\,;q \to p'\,;q} \quad \frac{p \xrightarrow{\checkmark}}{p\,;q \to q} \quad \frac{}{\texttt{while}\ (e)\ \texttt{do}\ \{p\} \to \texttt{if}\ (e)\ \texttt{then}\ \{p; \texttt{while}\ (e)\ \texttt{do}\ \{p\}\}\ \texttt{else}\ \{\texttt{skip}\}}$$

and other familiar rules for the remaining operators $*, \neg, =, <=, \wedge$ (see e.g. [13]). It can easily be verified that these rules induce a natural transformation $\rho : \Sigma(X \times BX) \to BT_\Sigma X$ that distributes syntax over behaviour. To illustrate, we define ρ for some operators. We suppose $X = (N, E, P)$ and abbreviate $\texttt{if}\ (e)\ \texttt{then}\ \{p\}\ \texttt{else}\ \{q\}$ to $\mathsf{if}(e, p, q)$, and similarly for $\mathsf{while}(e, p)$:

$$\begin{array}{lll}
+((u, b_u), (v, b_v)) & \longmapsto \mathsf{Cases}\{ & b_u = \underline{n} \text{ in } \mathbb{N} : +_n(v), b_u = u' \text{ in } N : (u' + v)\} \\
+_n((v, b_v)) & \longmapsto \mathsf{Cases}\{ & b_v = \underline{m} \text{ in } \mathbb{N} : (\underline{n+m}), b_v = v' \text{ in } N : +_n(v')\} \\
\texttt{if}\ ((e, b_e), (p, b_p), (q, b_q)) & \longmapsto \mathsf{Cases}\{ & b_e = \underline{false} \text{ in } \mathbb{B} : (q),\quad b_e = \underline{true} \text{ in } \mathbb{B} : (p), \\
 & & \quad b_e = e' \text{ in } E : (\texttt{if}(e', p, q))\} \\
\texttt{while}\ ((e, b_e), (p, b_p)) & \longmapsto \texttt{if}(e, & (p\,;\texttt{while}(e, p)),\ \texttt{skip})
\end{array}$$

Theorem 5.1 of [24] shows how ρ induces operational models by structural recursion. with closed programs being the carrier of the model.

Proposition 2.6 *Suppose $\rho_X : \Sigma(X \times BX) \to BT_\Sigma X$ is a natural transformation. For every coalgebra $\langle Y, \gamma : Y \to BY\rangle$, there is a unique morphism, which we denote $\tilde{T}\gamma$, such that the following diagram commutes. Here, η and μ are the unit and multiplication of T, and ψ is the Σ-algebra structure of TY.*



To induce the desired operational model – a coalgebra structure for closed programs, $T0 \to BT0$ – we take $Y = 0$ and $\gamma = ?_{BY} : 0 \to BY$ the initial map.

Example 2.7 For Example 2.5, this operational model describes the transition behaviour of while programs without variable lookup or update. Coalgebraic bisimilarity is given by a map from $T_\Sigma 0$ into the final B-coalgebra, which is $(\mathbb{N} \times \mathbb{N}, \mathbb{N} \times \mathbb{B}, \mathbb{N} \times 1)$ – it maps expressions of each type to the number $n \in \mathbb{N}$ of steps-to-termination and the terminal value $v \in \mathbb{N}, \mathbb{B}, 1$ for that expression. Two expressions p, q are then behaviourally equivalent iff they terminate in the same number of steps, and with the same terminal value.

2.1 Effects as Syntax

We now extend an abstract operational semantics with effectful commands, such as variable lookup/update for while programs. We extend the signature by adding constructors that describe effects such as variable lookup at every sort.

Definition 2.8 *An* effect signature *is a single-sorted signature* Eff *representing effects of arity at most* ω. *The* effectful extension $\mathsf{Sig} \oplus \mathsf{Eff}$ *of a language* Sig *with effects* Eff *consists of the function symbols in* Σ, *together with a function symbol* $\delta_s : (s)_{0 \le j < \alpha} \to s$ *(i.e. all arguments of type* s*) for each* $\delta \in \mathsf{Eff}$ *of arity* α *and all sorts* s *of* Sig. *If* $\Delta : \mathcal{C} \to \mathcal{C}$ *is the signature functor induced by* Eff, *we write* $\tilde{\Delta}$ *for* $\Delta^n : \mathcal{C}^n \to \mathcal{C}^n$. *Thus* $\Sigma + \tilde{\Delta}$ *is the signature functor induced by* $\mathsf{Sig} \oplus \mathsf{Eff}$.

Key examples of effects include global state, interactive I/O, and non-determinism. We focus on global state [15] as required for while programs.

Example 2.9 Given a finite set of variable-locations L with values in V, the effects for global state are 'read' $\mathrm{rd} : \mathsf{v} \to l$ and 'write' $\mathrm{wr} : 1 \to l \times \mathsf{v}$ operations on variables, where $|L| = l < \omega$ and $|V| = v \le \omega$. Thus the effect signature Eff has function symbols rd_x of arity v, and $\mathrm{wr}_{x,n}$ of arity 1, for all $x \in L$, $n \in V$.

We read $\mathrm{rd}_x((c_n)_{n \in V})$ as the computation that reads the value n of x in the store and then executes the command c_n. The command $\mathrm{wr}_{x,n}(c)$ assigns n to x and proceeds with command c.

The free-algebra functor $T_{\Sigma + \tilde{\Delta}}$ (Remark 2.3) describes the program terms of the extended language, allowing effects to be freely incorporated into program syntax. In particular, an extension of stateless while with effects for global state allows us to describe 'standard' while-programs in the following way.

Example 2.10 Extending the signature Sig_{sl} for *stateless while* with Eff given by global state introduces new syntax operators rd_x and $\mathrm{wr}_{x,n}$ of arity ω and 1 respectively (for all $x \in L$ and $n \in \mathbb{N}$), i.e. $\Delta X = L \times X^\omega + L \times \mathbb{N} \times X$ where L is the

set of store locations. We can thus write numeric expressions like $3+\mathrm{rd}_x(0,1,2,\ldots)$ – corresponding to $3+x$ – and we can express $x{:=}3$ as $\mathrm{wr}_{x,3}(\mathtt{skip})$.

2.2 Effectful Behaviour and the Final Coalgebra in Kl(M)

We argue that effects are most naturally captured by moving from the underlying category $\mathcal{C}^{\mathbb{S}}$ to a Kleisli category $\mathsf{Kl}(M)$ for a suitable monad M, where MX constructs an appropriate set of effect-trees with leaves in X. As described in [8], program execution can be understood as producing a syntactic effect-tree whose leaves are programs. If M constructs these effect-trees, then a suitable functor for effectful behaviour is MB, where B is as before. An effectful operational model is thus an MB-coalgebra with carrier $T_{\Sigma+\tilde{\Delta}}0$.

This structure is appropriate for describing the one-step evolution of programs as they are executed. However, it does not directly yield the right notion of multi-step evaluation; iterating a coalgebra map $\gamma : X \to MBX$ in the underlying category $\mathcal{C}$ gives a map $X \xrightarrow{\gamma} MBX \xrightarrow{MB\gamma} MBMBX$ that fails to accumulate the effects. However, a distributive law $\lambda : BM \to MB$ allows us to accumulate effects correctly as follows:

$$X \xrightarrow{\gamma} MBX \xrightarrow{MB\gamma} MBMBX \xrightarrow{M\lambda_{BX}} M^2B^2X \xrightarrow{\mu_{B^2X}} MB^2X \tag{1}$$

The effects arising in the first two execution steps have been combined, giving a single effect tree whose leaves (in B^2X) describe two effectless transition steps.

This result can be achieved naturally by moving from $\mathcal{C}$ into a Kleisli category $\mathsf{Kl}(M)$; MB-coalgebras in $\mathcal{C}$ are interpreted as $\overline{B}$-algebras in $\mathsf{Kl}(M)$ for a behaviour functor $\overline{B}$ obtained by 'lifting' B from $\mathcal{C}$ into $\mathsf{Kl}(M)$. The above chain of morphisms then corresponds to the iteration $\overline{B}\gamma \circ \gamma$; we can think of Kleisli-morphisms as accumulating and propagating effects.

Recall the objects of the Kleisli category $\mathsf{Kl}(M)$ are the same as the underlying category $\mathcal{C}$, but with morphisms $f' : X \to Y$ in 1-1 correspondence with the ('underlying') morphisms $f : X \to MY$ in $\mathcal{C}$. Composition $g' \circ f'$, for $g' : Y \to Z$ and $f' : X \to Y$, is given by the arrow $h' : X \to Z$ corresponding to the morphism $X \xrightarrow{f} MY \xrightarrow{Mg} MMZ \xrightarrow{\mu_Z} MZ$ (its 'overlying' arrow). We write $g^\dagger$ for the latter part, $\mu_Z \circ Mg : MY \to MZ$, and use notation $'$ in the natural way – e.g. the above identity becomes $g' \circ f' = (g^\dagger \circ f)'$.

We write J for the canonical (left-adjoint) inclusion functor $\mathcal{C} \to \mathsf{Kl}(M)$, identity-on-objects and sending $f : X \to Y$ to $Jf = \eta_Y \circ f : X \to Y$ [3]. One finds that $g' \circ Jf = (g \circ f)'$ and $Jf \circ h' = (Mf \circ h)'$.

A 'lifting' $\overline{B}$ of the behaviour functor B into $\mathsf{Kl}(M)$ satisfies $JB = \overline{B}J$. In particular, this implies that $\overline{B}$ is identity-on-objects – so a Kleisli-morphism $\gamma' : X \to \overline{B}X$ has underlying type $\gamma : X \to MBX$, establishing the 1-1 correspondence between MB-coalgebras and $\overline{B}$-coalgebras mentioned above.

Liftings can be described in terms of distributive laws [21,3]:

Lemma 2.11 *There is a 1-1 correspondence between liftings $\overline{B}$ of B into $\mathsf{Kl}(M)$ and distributive laws $\lambda : BM \to MB$. For B defined by sums and copowers (i.e. isomorphic to $BX = V + A \times X$ for some V, A), these distributive laws exist.*

Given a distributive law λ, the corresponding $\overline{B}$ sends an arrow $f' : X \to Y$ to the arrow $\overline{B}f' : BX \to BY$ corresponding to $BX \stackrel{Bf}{\to} BMY \stackrel{\lambda_Y}{\to} MBY$. Thus for $\gamma' : X \to \overline{B}X$, the composition $\overline{B}\gamma' \circ \gamma'$ corresponds to the chain of arrows anticipated in (1):

$$(\overline{B}\gamma)^\dagger \circ \gamma \;=\; (\lambda_X \circ B\gamma)^\dagger \circ \gamma \;=\; \mu_{BX} \circ M\lambda_X \circ MB\gamma \circ \gamma$$

We later use the easy fact that if f is an MB-coalgebra morphism, then Jf is a $\overline{B}$-coalgebra morphism.

The terminal sequence for $\overline{B}$ in the Kleisli category suggests that the final $\overline{B}$-coalgebra in $\mathsf{Kl}(M)$, if it exists, is a natural candidate for program denotations. Standard results guaranteeing its existence are reviewed below [21,4,7].

Definition 2.12 *A category $\mathcal{C}$ is* Cppo-enriched *if all hom-sets are complete partial orders with least element $\bot$ and composition is ω-continuous in both arguments ($g \circ \bigsqcup_n f_n = \bigsqcup_n (g \circ f_n)$ and similarly for the first argument). Composition in $\mathcal{C}$ is* left-strict *if $\bot_{X,Y} \circ f = \bot_{X,Z}$ for $f : Y \to Z$. A functor $B : \mathcal{C} \to \mathcal{D}$ between* Cppo-*enriched categories $\mathcal{C}, \mathcal{D}$ is* locally continuous *if it preserves suprema of ω-chains, i.e. $B(\bigsqcup_n f_n) = \bigsqcup_n B(f_n)$, and* locally monotone *if $Bf \leq Bg$ whenever $f \leq g$.*

The following is shown in [4, Theorem 3.3 and Proposition 3.9] and readily extends componentwise to the multi-sorted setting.

Proposition 2.13 *Let M be a monad on $\mathcal{C}$, $\mathsf{Kl}(M)$ a Cppo-enriched Kleisli category with left-strict composition, and $\overline{F}$ a locally-continuous lifting of F into $\mathsf{Kl}(M)$. If the initial F-algebra $\langle \overline{D}, \alpha \rangle$ exists in $\mathcal{C}$, then the final $\overline{F}$-coalgebra in $\mathsf{Kl}(M)$ is given by $\langle \overline{D}, J\alpha^{-1} \rangle$. If $\mathcal{C} = \mathsf{Set}^n$, F need only be locally monotonic.*

Proof. Follows from [4] Propositions 3.2 and 3.9. Note that in Proposition 3.2 the initial F-algebra in $\mathsf{Kl}(M)$ is given by $\eta \circ \alpha$, where η is the unit of M; its inverse in $\mathsf{Kl}(C)$ is easily shown to be $\eta \circ \alpha^{-1}$, or equivalently $J\alpha^{-1}$. □

Particular effect monads M may satisfy these constraints, or be modified to do so. However, ideally one would like to generate such monads from a given effect signature. We must equip the Kleisli category with a Cppo-enriched structure with left-strict composition; obtaining appropriate $\overline{B}$ is usually not a problem. For a given effect signature Δ without equations, defining a Cppo-enriched structure is not problematic. In Set^n, we may define:

Definition 2.14 *For a countably-polynomial functor $\tilde{\Delta}$ on $\mathcal{C}^n$, we define $F_{\tilde{\Delta}}$ to be the cofree $\tilde{\Delta} + \bot$-coalgebra functor on $\mathcal{C}^n$ where $\bot \cong 1$.*

$F_{\tilde{\Delta}}$ is indeed a monad (Theorems 2 and 4 of [11]), and is given component-wise by F_Δ: $(F_{\tilde{\Delta}}X)_s = F_\Delta(X_s)$. The objects $F_{\tilde{\Delta}}$ may be given the natural partial order structure. In Cppo, this structure appears as the initial $\tilde{\Delta}$-algebra [2].

By contrast, left-strictness is not so straightforward. Intuitively, this requires that if we replace all leaves of an effect-tree with $\bot$, the resulting tree is identified with $\bot$. We anticipate a better solution in Cppo, perhaps by introducing divergence as an effect ([5] Example 6); for now, we give a strategy in Set for adjusting $F_{\tilde{\Delta}}$ in Set to ensure left-strictness.

2.2.1 Left-strictness in Set

In Set, we may achieve left-strictness by restricting $F_\Delta X$ as follows:

Definition 2.15 *We write G_Δ for the functor with $G_\Delta X \subseteq F_\Delta X$ consisting of the effect-trees which do not have a non-trivial subtree with all leaves equal to $\bot$. On morphisms, $G_\Delta f$ is the restriction of $F_\Delta f$. We write $G_{\tilde{\Delta}}$ for G_Δ applied componentwise in Set^n.*

It is straightforward to show that G_Δ is a functor on Set. An ordering $\sqsubseteq_G$ on $G_\Delta X$ is inherited from that of $F_\Delta X$: for $t, t' \in G_\Delta X$, $t \sqsubseteq_G t'$ iff t' is obtained from t by replacing some $\bot$ leaves with subtrees. G_Δ is easily defined on arrows $f : X \to Y$ as the restriction of $F_\Delta f : F_\Delta X \to F_\Delta Y$ to $G_\Delta X$, which simply relabels each leaf $x \in X$ with $f(x)$. (The result is a tree in $G_\Delta Y$.)

G_Δ has a monad structure similar to that of F_Δ: the unit η_X maps $x \in X$ to the singleton tree with leaf x (which lies in $G_\Delta X$); the multiplication $\mu_X : G_\Delta^2 X \to G_\Delta X$ 'plugs in' the trees at each leaf and then prunes any resulting all-$\bot$ subtrees. The monad axioms are straightforward to verify, and the definition of multiplication ensures that $\bot^\dagger = \mu \circ G_\Delta \bot$ indeed maps every tree to $\bot$, so that $\mathsf{Kl}(G_\Delta)$ is left-strict.

We have that $G_\Delta X \subseteq F_\Delta X$ is a sub-cppo of $F_\Delta X$, as the limit of an ω-chain in $G_\Delta X$ also lies in $G_\Delta X$. This follows from the contrapositive: if the limit t_ω of an ω-chain (t_n) in $F_\Delta X$ has a non-trivial all-$\bot$ subtree ($t_\omega \notin G_\Delta X$) then some t_n must also have non-trivial all-$\bot$ subtree ($t_n \notin G_\Delta X$).

Thus $\mathsf{Kl}(G_\Delta)$ has a Cppo-enriched structure, obtained pointwise from that given by G_Δ. To apply proposition 2.13 for behaviour functor $BX = V + A \times X$, it remains to supply a locally monotonic lifting $\overline{B}$, or equivalently a distributive law $\lambda_X : BG_\Delta X \to G_\Delta BX$. This may be defined as before: $\lambda_X(\mathsf{inl}(v)) = \eta_X(\mathsf{inl}(v))$, and $\lambda_X(\mathsf{inr}(a,t)) = t'$ where t' is obtained by replacing every non-$\bot$ leaf x with $\mathsf{inr}(a,x)$. Local monotonicity follows from the fact that for an arrow $f : X \to G_\Delta Y$, $\overline{B}f = \lambda_Y \circ Bf$ takes as input either a value $\mathsf{inl}(v)$ or a pair $(a,x) \in A \times X$; it correspondingly returns the singleton tree with leaf $\mathsf{inl}(v)$, or the tree obtained from $f(x) \in G_\Delta Y$ by replacing every non-$\bot$ leaf y with (a,y).

Corollary 2.16 *There is a lifting of $BX = V + A \times X$ to $\mathsf{Kl}(G_\Delta)$, and $\mathsf{Kl}(G_\Delta)$ has a final $\overline{B}$-coalgebra as given by Proposition 2.13.*

Lastly, this result extends componentwise to $G_{\tilde{\Delta}}$ on Set^n.

3 From Effectless to Effectful Abstract Operational Semantics

Given a suitable monad M, we consider the problem of obtaining a suitable operational model, an MB-coalgebra with carrier $T_{\Sigma+\tilde{\Delta}}0$, the set of closed program syntax terms mixed with effects. Theorem 2.6 allows us to do this via structural recursion; it requires an operational specification – an abstract operational semantics – for the effectfully extended language with syntax functor $\Sigma + \tilde{\Delta}$ and behaviour M, i.e. a natural transformation $\rho^{\mathsf{Eff}} : (\Sigma + \tilde{\Delta})(X \times MBX) \to MBT_{\Sigma+\tilde{\Delta}}X$.

This section describes a first attempt at systematically obtaining ρ^{Eff} from an abstract operational semantics ρ for the *effectless* fragment of the language. Note that one does not have to use ρ; one could directly specify ρ^{Eff} for the language in question, avoiding the need for dependency-functions below, but the operational rules become more intricate; we hope to elaborate on this in future work.

Remark 3.1 *In addition to Proposition 2.13, we assume M is given componentwise by a monad M_0, and has a (componentwise) strength and a natural transformation $\phi_X : \tilde{\Delta}MX \to MX$. In* Set *this holds for $M = G_{\tilde{\Delta}}X$, using the $\tilde{\Delta}$-part of the inverse of the $\tilde{\Delta} + \bot + X$ coalgebra structure of $F_{\tilde{\Delta}}X$.*

To define ρ^{Eff} from ρ, we must handle program syntax Σ and effect syntax $\tilde{\Delta}$. One readily obtains a natural transformation (later called ψ^1) for the latter, but it is less easy to define suitable $\psi^2_X : \Sigma(X \times MBX) \longrightarrow MBT_\Sigma X$ to handle existing syntax. We soon show how an assumption that ρ is 'dependency-supported' allows us to define ψ^2 correctly.

Let us attempt to describe the effectful behaviour of `if` (e) `then` $\{p\}$ `else` $\{q\}$ when e, p, q exhibit effectful behaviour. For instance, e might have behaviour $\mathrm{rd}_x(\underline{true}, e', \underline{false}, \ldots)$ – depending on whether x is $0, 1, 2$, e terminates as $\underline{true}$, evolves to e', terminates as $\underline{false}$, and so on. We would expect the behaviour of `if` (e) `then` $\{p\}$ `else` $\{q\}$ to depend similarly on the value of x – it would respectively evolve to p, `if` (e') `then` $\{p\}$ `else` $\{q\}$, q, and so on. Note that we never consider the behaviour of p or q, effectful or otherwise; effects present in their behaviour should only be considered once the condition e is evaluated.

Now we consider how to formalise this in Set^3. Assume a tuple $X = (N, E, P)$ with $e \in E$ and $p, q \in P$. The possible behaviours of e and p, q are the second and third components of $BX = (N + \mathbb{N}, E + \mathbb{B}, P + 1)$. Recall that effectless operational rules are commonly represented by a natural transformation $\Sigma(X \times BX) \to BT_\Sigma X$. For if statements, this gives a function $\rho : (E \times BE) \times (P \times BP)^2 \to BT_\Sigma X$ whose arguments are respectively a pair (e, b) of a boolean variable e and an (effectless) behaviour for it, and similarly for programs $p, q \in P$.

Suppose we now have effectful behaviours for e, p, q – elements of MBE and MBP, rather than BE and BP. We aim to define a function analogous to ρ, but incorporating effectful behaviour: $(E \times MBE) \times (P \times MBP)^2 \longrightarrow MBT_\Sigma X$. The informal argument involves 'pulling out' the effects present in the behaviour of e,

which corresponds to two applications of monad strength [12]:

$$(E \times MBE) \times (P \times MBP)^2 \longrightarrow M((E \times BE) \times (P \times MBP)^2)$$

However, now we are stuck. We need something of form $(E \times BE) \times (P \times BP)^2$ so that we can apply ρ; however, repeatedly using the monad strength would propagate the behaviours of p and q, which is incorrect as argued above. At this point, the informal argument used the fact that the behaviours of p and q are irrelevant. Thus one might simply *discard* the effectful behaviours MBP and replace them with arbitrary values in BP:

$$M((E \times BE) \times (P \times MBP)^2) \to M((E \times BE) \times (P \times BP)^2)$$

Now applying $M\rho$ to `if` (e) `then` $\{p\}$ `else` $\{q\}$ gives the correct effectful behaviour. However, it is a crude way to exploit the information that 'the behaviour of `if` (e) `then` $\{p\}$ `else` $\{q\}$ does not depend on that of p or q'. We express this dependency more systematically, assuming that ρ has an appropriate factorisation – deducible from the operational rules – discarding irrelevant arguments. Without loss of generality, we assume the first n arguments must be kept.

This gives a canonical means of defining ψ^2 if the behaviours of every program term depends on that of *at most one subterm* – i.e. operational rules have at most one premise. However, two issues arise in the presence of multiple premises. Consider a 'synchronous execution' operator:

$$\frac{p \to p', q \to q'}{p \times q \to p' \times q'} \qquad \frac{p \to p', q \overset{\surd}{\to}}{p \times q \to p'} \qquad \frac{p \overset{\surd}{\to}, q \to q'}{p \times q \to q'} \qquad \frac{p \overset{\surd}{\to}, q \overset{\surd}{\to}}{p \times q \overset{\surd}{\to}}$$

If p and q both introduce variable updates (e.g. to the same variable), obviously there can be no canonical choice for the effectful behaviour of $p \times q$; one must make a choice whether to apply the variable update of p first, or of q. In the presence of equations, if the effects are commutative (e.g. non-determinism), this choice makes no difference and the extension is canonical; otherwise one is forced to put an ordering on the sub-terms specifying the order of propagation. Without loss of generality, we will suppose this ordering is simply left-to-right, and define a natural transformation `comb` to propagate effects in this way using monadic strength.

The second issue is that a term may not always depend on the same number of sub-terms. In while, $+$ and $*$ are an example of this; without auxiliary operators $+_n$ and $*_n$, a standard operational semantics would contain the rules

$$\frac{u \to u'}{u + v \to u' + v} \qquad \frac{u \overset{\surd}{\to} \underline{n}, v \to v'}{u + v \to u + v} \qquad \frac{u \overset{\surd}{\to} \underline{n}, v \overset{\surd}{\to} \underline{m}}{u + v \overset{\surd}{\to} \underline{n + m}}$$

In applying the first rule, we must not propagate any effects of v, unlike the other cases. This would require a more fine-grained approach than the one above; however, introducing auxiliaries can ensure the behaviour of every syntax constructor once

again always depends on the same sub-terms – here, we achieve this by introducing $+_n$ and $*_n$ as above.

For the rest of this section, we will assume that the behaviour of every program term of arity α depends on some number $n \leq \alpha$ of subterms, whose effects are to be propagated from left-to-right. We make this formal:

Definition 3.2 *For a signature* Sig*, a dependency function dep* : $\mathsf{Sig} \to \mathbb{N}$ *is a function satisfying* $0 \leq dep(f) < \mathrm{ar}(f)$ *for every symbol* $f \in \mathsf{Sig}$.

Definition 3.3 *Given a signature* Sig *with corresponding syntax functor* Σ *and a dependency function dep, for each* Y *in* $\mathcal{C}^{\mathbb{S}}$*, the restriction* $\mathsf{res}_{Y,X}$ *is a natural transformation in* X*, whose* s*-component is given by:*

$$\begin{array}{rll} \mathsf{res}_{Y,X}: & (\Sigma X)_s = & \left(\coprod_{f:(s_i)\to s\ \in \mathsf{Sig}} \left(\prod_{0\leq i<\mathrm{ar}(f)} (X\times Y)_{s_i}\right)\right)_s \\ & \xrightarrow{\cong} & \left(\coprod_{f:(s_i)\to s\ \in \mathsf{Sig}} \left(\prod_{0\leq i<\mathrm{ar}(f)} Y_{s_i} \times \prod_{0\leq i<\mathrm{ar}(f)} X_{s_i}\right)\right)_s \\ & \xrightarrow{\coprod(\prod \pi_i\times \mathsf{id})} & \left(\coprod_{f:(s_i)\to s\ \in \mathsf{Sig}} \left(\prod_{0\leq i<dep(f)} Y_{s_i} \times \prod_{0\leq i<\mathrm{ar}(f)} X_{s_i}\right)\right)_s \end{array}$$

When $dep(f) = 0$*, we take* π_0 *as the terminal map and* Y_{s_0} *as the initial object* 1.

Definition 3.4 *Given a dependency function dep, an abstract operational semantics* $\rho_X : \Sigma(X \times BX) \to BT_\Sigma X$ *is* dep-supported *if there is a natural transformation* ρ' *such that* ρ_X *is given by the composition*

$$\Sigma X \xrightarrow{\mathsf{res}_{BX,X}} \left(\coprod_{f:(s_i)\to s} \left(\prod_{0\leq i<dep(f)} (BX)_{s_i} \times \prod_{0\leq i<\mathrm{ar}(f)} X_{s_i}\right)\right)_s \xrightarrow{\rho'_X} BT_\Sigma X$$

Trivially, every language is dep-supported if we set $dep(f) = \mathrm{ar}(f)$ for all f; but this will not necessarily give the desired behaviour, as illustrated above.

Example 3.5 For while programs, only the behaviour of the first argument matters – except for `while` statements, and constants, which have no arguments. Thus we define a dependency function by $dep(f) = 1$ for all f except the constants ($n \in \mathbb{N}, b \in \mathbb{B}$, and `skip`), and `while` (e) `do` $\{p\}$ – which have $dep(f) = 0$. It is easily shown that the abstract operational semantics for `While` is then dep-supported.

To handle multiple premises, one needs a method comb of combining effect-trees. One may use the monad strength $\mathsf{st}_{X,Y}$, of type $X \times (MY) \longrightarrow M(X \times Y)$. Here, it takes an x and an effect-tree with leaves in Y, and pairs each leaf y with x, giving an effect-tree with leaves in $X \times Y$.

Definition 3.6 *For all* $n < \omega$*, given objects* $Y_0, \ldots, Y_{n-1}$ *we inductively define an arrow* $\mathsf{comb}_{Y_0,\ldots,Y_{n-1}} : \prod_{0\leq i\leq n-1} (MY_i) \longrightarrow M\left(\prod_{0\leq i\leq n-1} Y_i\right)$ *by* $\mathsf{comb}_{Y_0} = \mathsf{id}_{MY_0}$*, and* $\mathsf{comb}_{Y_0,\ldots,Y_{n+1}}$ *in terms of* $\mathsf{comb}_{Y_0,\ldots,Y_n}$ *as follows:*

$$\begin{array}{lll} & MY_n \times \prod_{0\le i<n} MY_i & \xrightarrow{\mathsf{id}\times\mathsf{comb}_{Y_0,\dots,Y_n}} & MY_n \times M\prod_{0\le i<n} Y_i \\ \xrightarrow{\mathsf{st}} & M(MY_n \times \prod_{0\le i<n} Y_i) & \xrightarrow{\cong} & M(\prod_{0\le i<n} Y_i \times MY_n) \\ \xrightarrow{M\mathsf{st}} & M^2\prod_{0\le i<n+1} Y_i) & \xrightarrow{\mu} & M\prod_{0\le i<n+1} Y_i \end{array}$$

3.1 Abstract Operational Semantics for Effectful Behaviour

Now we define the ρ^{Eff} from the beginning of this section, assuming ρ is *dep*-supported. We express it in terms of the following components:

$$\psi^1_X : \tilde{\Delta}(X \times MBX) \to MBT_\Sigma X \qquad \psi^2_X : \Sigma(X \times MBX) \to MBT_\Sigma X$$

We may then define $\rho^{\mathsf{Eff}} = MB\,\mathsf{inc} \circ [\psi^1_X, \psi^2_X]$ where inc 'includes' terms given by $T_\Sigma X$ into the 'bigger language' $T_{\Sigma+\tilde{\Delta}}X$. We can do this by giving $T_{\Sigma+\tilde{\Delta}}X$ the evident Σ-algebra structure; then inc is the initial Σ-algebra map $T_\Sigma X \to T_{\Sigma+\tilde{\Delta}}X$.

Definition 3.7 *We define ψ^1_X by $MB\eta_X \circ \phi_{BX} \circ \tilde{\Delta}\pi_2$, where π_2 is the projection $X \times Y \to Y$, ϕ_{BX} is as given by Remark 3.1, and η is the unit of T_Σ.*

Definition 3.8 *ψ^2_X is defined below. In the second map, ζ isomorphically replaces $(MBX)_{s_i}$ with $M_0((BX)_{s_i})$ (Remark 3.1) if $dep(f) > 0$, otherwise it is the unit $\eta_1 : 1 \to M1$ (recall $(-)_{s_0} = 1$ by Definition 3.3). The fourth uses strength of T_Δ. The fifth map 'swaps the M and the coproduct $\coprod_{f\in\mathsf{Sig}}$' as follows. For each f we write inj_f for the injection into the f-component of the coproduct $Y \to \coprod_{g\in\mathsf{Sig}} Y_g$; we apply M to inj_f and take the coproduct $[M\mathsf{inj}]_f$ over all $f \in \mathsf{Sig}$.*

$$\begin{array}{ll} & \psi^2_X : \quad \Sigma(X \times MBX) \\ \xrightarrow{\mathsf{res}_{(MBX),X}} & \left(\coprod_{f:(s_i)\to s\in\mathsf{Sig}} \left(\prod_{0\le i<depf} (MBX)_{s_i} \times \prod_{0\le i<\mathrm{ar}(f)} X_{s_i}\right)\right)_s \\ \xrightarrow{(\coprod_f(\zeta\times\mathsf{id}))_s} & \left(\coprod_{f:(s_i)\to s} \left(\prod_{0\le i<depf} M(BX)_{s_i} \times \prod_{0\le i<\mathrm{ar}(f)} X_{s_i}\right)\right)_s \\ \xrightarrow{(\coprod_f(\mathsf{comb}\times\mathsf{id}))_s} & \left(\coprod_{f:(s_i)\to s} \left(M\left(\prod_{0\le i<depf} (BX)_{s_i}\right) \times \prod_{0\le i<\mathrm{ar}(f)} X_{s_i}\right)\right)_s \\ \xrightarrow{(\coprod_f \mathsf{st})_s} & \left(\coprod_{f:(s_i)\to s} \left(M\left(\prod_{0\le i<depf} (BX)_{s_{dep(f)}} \times \prod_{0\le i<\mathrm{ar}(f)} X_{s_i}\right)\right)\right)_s \\ \xrightarrow{([M\mathsf{inj}_f]_{f\in\mathsf{Sig}})_s} & \left(M\left(\coprod_{f:(s_i)\to s} \left(\prod_{0\le i<depf} (BX)_{s_{dep(f)}} \times \prod_{0\le i<\mathrm{ar}(f)} X_{s_i}\right)\right)\right)_s \\ \xrightarrow{M\rho'} & MBT_\Sigma X \qquad \xrightarrow{M\mathsf{inc}} \qquad MBT_{\Sigma+\tilde{\Delta}}X \end{array}$$

Lemma 3.9 *ψ^1 and ψ^2 are natural transformations.*

Proof. All the components of the definition – namely $\phi_B X$, η_X, $\mathsf{res}_{Y,X}$, ρ'_X, inc_X, $\mathsf{comb}_{X_{s_1},\dots,X_{s_n}}$, and $\mathsf{st}_{BX,\coprod X_{s_i}}$ and inj_f for any $f \in \mathsf{Sig}$ – are natural in X. □

Thus we obtain ρ^{Eff} from Definition 3.7. It is natural because it is defined in terms of natural transformations ψ^1, ψ^2, and $\mathsf{inc}_X : T_\Sigma X \to T_{\Sigma+\tilde{\Delta}}X$.

Example 3.10 We describe the action of ρ^{Eff} for stateless while programs extended with global state. It takes as input terms of form $\sigma((x_1,t_1),(x_2,t_2),\dots)$ where σ is

either a syntax constructor for stateless while (Example 2.2), or effect syntax for global state, viz. rd_x or $\mathrm{wr}_{x,n}$. Each x_i is a variable (of appropriate sort), and t_i a corresponding effectful behaviour – an effect-tree (of reads rd_x and updates $\mathrm{wr}_{x,n}$) whose leaves are variables y_i or terminal values v_i. The output is an element of $MBT_{\Sigma+\tilde{\Delta}}X$ – an effect-tree with each leaf either an expression t, or a terminal value v of the same sort as σ.

ρ^{Eff} acts straightforwardly (via ψ^1) on effect terms such as $\mathrm{wr}_{x,n}((p,t_p))$, where p is an expression and t_p an effectful behaviour. This term is simply mapped to $\mathrm{wr}_{x,n}(p)$; similarly, $\mathrm{rd}_x((x_1,t_1),(x_2,t_2),\ldots)$ is mapped to $\mathrm{rd}_x(x_1,x_2,\ldots)$.

The effect on syntax terms like $+((m,t_m),(n,t_n))$ follows from the definition of ψ^2. The restriction res discards the behaviour t_n as it is irrelevant; the strength comb 'pulls' the tree t_m out, giving an effect tree t'_m of the same shape as t_m, but whose leaves are of form $+((m,b_m),n)$ where b_m is a leaf of t_m, an effectless behaviour. Finally, the standard operational semantics (Example 2.5) is applied to each leaf of t'_m. As an illustration, we might have

$$+((n,\mathrm{rd}_x(n',5+2,\underline{3},\ldots)),(m,\underline{5})) \;\mapsto\; \mathrm{rd}_x(n'+m,+_5(2)+m,+_3(m),\ldots)$$

Given ρ^{Eff}, Theorem 2.6 yields an operational model by structural recursion:

Corollary 3.11 *Given an effect signature* Eff*, every dep-supported abstract operational semantics* $\rho : \Sigma(X \times BX) \to BT_\Sigma X$ *gives rise to an effectful operational model, an* MB*-coalgebra with carrier carrier* $T_{\Sigma+\tilde{\Delta}}0$*.*

Example 3.12 We illustrate the operational behaviour of the program

$$p \;=\; \texttt{if}\ (x=1)\ \texttt{then}\ \{y{:=}2\}\ \texttt{else}\ \{\texttt{skip}\}$$

where x is shorthand for $\mathrm{rd}_x(0,1,2,\ldots)$, and $y{:=}2$ shorthand for $\mathrm{wr}_{y,2}(\texttt{skip})$:

$$\begin{aligned}
&\longrightarrow \mathrm{rd}_x(\texttt{if}\ (0=1)\ \texttt{then}\ \{y{:=}2\}\ \texttt{else}\ \{\texttt{skip}\},\ \texttt{if}\ (1=1)\ \texttt{then}\ \{y{:=}2\}\ \texttt{else}\ \{\texttt{skip}\},\\
&\qquad\qquad \texttt{if}\ (2=1)\ \texttt{then}\ \{y{:=}2\}\ \texttt{else}\ \{\texttt{skip}\},\ldots))\\
&\longrightarrow \mathrm{rd}_x(\texttt{if}\ (=_0(1))\ \texttt{then}\ \{y{:=}2\}\ \texttt{else}\ \{\texttt{skip}\},\ \texttt{if}\ (=_1(1))\ \texttt{then}\ \{y{:=}2\}\ \texttt{else}\ \{\texttt{skip}\},\\
&\qquad\qquad \texttt{if}\ (=_2(1))\ \texttt{then}\ \{y{:=}2\}\ \texttt{else}\ \{\texttt{skip}\},\ldots))\\
&\longrightarrow \mathrm{rd}_x(*,\mathrm{wr}_{y,2}(\texttt{skip}),*,*,\ldots) \quad\longrightarrow\quad \mathrm{rd}_x(*,\mathrm{wr}_{y,2}(*),*,*,\ldots)
\end{aligned}$$

If B has a lifting to $\mathsf{Kl}(M)$, the operational model can be seen as a $\overline{B}$-coalgebra living in the Kleisli category. If in addition the final $\overline{B}$-coalgebra exists, we may thus define an operational equivalence for programs in terms of the unique Kleisli-coalgebra map $[\![-]\!]$ into the final $\overline{B}$-coalgebra. For $\mathcal{C}=\mathsf{Set}$ and $BX = V + A \times X$, Proposition 2.16 implies both these conditions.

Definition 3.13 For $\mathcal{C}=\mathsf{Set}$, two programs $p,q \in T_{\Sigma+\tilde{\Delta}}0$ are *operationally equivalent*, $p \cong_{op} q$, if $[\![p]\!] = [\![q]\!]$ where $[\![-]\!]$ is the unique $\overline{B}$-coalgebra morphism into the final M-coalgebra.

Example 3.14 Theorem 2.13 implies the carrier of the final $\overline{B}$-coalgebra is the initial B-algebra, $\overline{D}$. The underlying arrow therefore maps into $M\overline{D}$. For `While` expressions, $\overline{D}$ is $(\mathbb{N}\times\mathbb{N}, \mathbb{N}\times\mathbb{B}, \mathbb{N}\times 1)$; hence (at sort s) the elements of $(M\overline{D})_s$

are effect-trees with leaves in $(n, v) \in \overline{D}_s$. One may show (by uniqueness) that the final $\overline{B}$-coalgebra arrow ! maps each term to the overall effect-tree it produces during execution, with leaves $(n, v) \in \overline{D}$ describing executions – n is the number of steps before termination, and v the final value of that execution. For example, the program p of Example 3.12 is mapped to $\mathrm{rd}_x((3,*), \mathrm{wr}_{y,2}(4,*), (3,*), (3,*), \ldots)$.

4 Towards Adequacy for Effectful Operational Semantics

In the previous section, we argued that an operational model for an effectful language finds its natural place in the Kleisli-category $\mathsf{Kl}(M)$. This section is devoted to the problem of obtaining an adequate effectful denotational semantics.

To discuss adequacy, it is convenient to change notation. From now on, we relabel the syntax functor of the extended language, $\Sigma + \tilde{\Delta}$, simply to Σ (i.e. Σ now incorporates effectful commands directly). We now write T for the free Σ-algebra functor. In the same vein, we forget about the effectless abstract OS $\rho : \Sigma(X \times BX) \to BTX$ and relabel the effectful abstract OS ρ^{Eff} to ρ.

The effectful denotational model arises by mapping the operational model into the final coalgebra $\overline{D}$ in the Kleisli category $\mathsf{Kl}(M)$. In the underlying category this gives a map $T0 \to M\overline{D}$, so we aim to take $M\overline{D}$ as our denotational model.

4.1 A Couple of Counterexamples

There is a key difference between the effectful and effectless settings: for some effectful operational specifications (abstract OS), the operational semantics is *not* compositional, and adequacy fails.

Example 4.1 Consider an interleaving operator | or a 'partial' (one-step) evaluator :>, defined by the following rules (at any type):

$$\frac{x \to x'}{x \mid y \to y \mid x'} \qquad \frac{x \overset{\checkmark}{\to} \underline{v}}{x \mid y \to y} \qquad \frac{x \to x'}{x :> y \to y} \qquad \frac{x \overset{\checkmark}{\to} v}{x :> y \to y}$$

These rules are single-premise, so they may be effectfully extended as in the previous section. Operationally, the effectful extension $x \mid y$ branches according to the effects given by the *first step* of behaviour of x, then by that of y; then more effects are introduced by the successors of x, then by the successors of y, and so on. $x :> y$ exhibits effects from the *first* step of x's execution only.

Now consider the following two programs, $p_1 = \mathrm{wr}_{y,1}(\texttt{skip}; \texttt{skip})$ and $p_2 = \texttt{skip}; \mathrm{wr}_{y,1}(\texttt{skip})$. They are identified by the map into the final Kleisli coalgebra (as they both assign $y = 1$ and terminate in 3 steps); thus are operationally equivalent. Yet if we put them in the contexts $[-] \mid q$ or $[-] :> q$, where

$$q = \mathrm{rd}_y(\mathrm{wr}_{z,0}(\texttt{skip}), \mathrm{wr}_{z,42}(\texttt{skip}), \mathrm{wr}_{z,42}(\texttt{skip}), \ldots)$$

the results are not operationally equivalent; if y is initially 0, then $p_1 \mid q$ will assign $z = 42$, and $p_2 \mid q$ will assign $z = 0$. $p_1 :> q$ and $p_2 :> q$ are similarly different. Thus, the operational semantics is not compositional with respect to $\mid$ or $:>$.

Compositionality fails because our denotational semantics intentionally discards information about precisely *when* effects occur during program execution. If the operational semantics of a syntax operator like $\mid$ depends on this information, our denotational semantics clearly cannot be adequate.

Thus we must restrict ρ to forbid too-fine-grained operational interaction between subterms. One option is a form of *evaluation-in-context*, where a subterm x_i is evaluated until all its execution branches have terminated, before evaluating any other terms on those branches. We formalise this notion below.

4.2 Extending The Original Semantics

We review the adequacy proof of Turi and Plotkin in our effectful setting, in terms of Σ-algebras and MB-coalgebras; our denotational and operational semantics arise as a quotient of theirs.

We write $\langle D, s\rangle$ for the final MB-coalgebra – the denotational model in Turi and Plotkin's original setting. We say D is a layered semantics, in that it describes all possible layerings of effects M interleaved with effectless transitions B.

Recall that $\langle \overline{D}, \alpha\rangle$ is the initial B-algebra. As before, we assume Proposition 2.13 holds for $F = B$, so that $\langle \overline{D}, J\alpha^{-1} : \overline{D} \to MB\overline{D}\rangle$ is the final $\overline{B}$-coalgebra. We equip the new denotational model $M\overline{D}$ with an MB-coalgebra structure, $\overline{s} = MB\eta^M \circ J\alpha^{-1}$. Like the final MB-coalgebra, this formally describes sequences of effect-layers M and effectless transitions B, except that every effect layer after the first is trivial (non-branching), as they arise from the unit η^M. In a sense, the effects are 'collected' into the first step.

By Proposition 2.6, ρ induces MB-coalgebra structures $\langle TD, \tilde{T}s\rangle$ and $\langle TM\overline{D}, \tilde{T}\overline{s}\rangle$ for syntax terms over the 'layered' and 'collected' denotational models respectively. By taking the syntax functor to be Σ, the behaviour functor to be MB, and the abstract OS to be ρ, we obtain the following diagram (cf. [24], p.84).

$$\begin{array}{ccc}
\Sigma T0 & \xrightarrow{\Sigma den} & \Sigma D \\
{\scriptstyle \psi_{T0}}\downarrow & & \downarrow{\scriptstyle \beta} \\
T0 & \xrightarrow[op]{den} & D \\
{\scriptstyle \tilde{T}?}\downarrow & & \downarrow{\scriptstyle s} \\
MBT0 & \xrightarrow{MBop} & MBD
\end{array}$$

where we write ψ_{T0} for the Σ-algebra structure of free-syntax ters $T0$. Proposition 2.6 induces (via ρ) the coalgebra structure $\tilde{T}?$ on closed terms $T0$, where $? : 0 \to MBT0$. The Σ-algebra structure β on the denotational model $M\overline{D}$ is obtained from the final MB-coalgebra morphism β_{TD} from $\langle TD, \tilde{T}s\rangle$ into D, as follows: $\beta : \Sigma D \xrightarrow{\Sigma\eta^T} \Sigma TD \xrightarrow{\psi_D} TD \xrightarrow{\beta_{TD}} D$.

The initiality of $T0$ induces the Σ-algebra morphism den into $\langle D, \beta \rangle$, giving a (necessarily) compositional map into the denotational model. op is the MB-coalgebra morphism from $T0$ into D induced by finality of the latter, giving an operational characterisation of program behaviour.

Turi and Plotkin's work implies that the operational map op is a Σ-algebra morphism, so that the initiality of $T0$ implies that $den = op$ (as there is only one such map), so that the operational and denotational maps coincide. In particular, it implies compositionality of the operational equivalence $\cong_{op}$ induced by op, as defined in 3.13 – i.e. $\cong_{op}$ is a congruence with respect to syntax given by Σ. It also implies that the denotational semantics is adequate with respect to $\cong_{op}$.

4.3 An Adaptation to the Kleisli Setting

Our strategy is to map from the layered denotational model D into $M\overline{D}$. As the former is an MB-coalgebra – and thus a $\overline{B}$-coalgebra – we obtain a unique $\overline{B}$-coalgebra morphism $c' : D \to \overline{D}$ in $\mathsf{Kl}(M)$ which effectively collects all the effects from every step of the behaviours in D into the first step, giving a single layer of effects and allowing the former diagram to be extended:

$$\begin{array}{ccccc}
\Sigma T0 & \xrightarrow{\Sigma op} & \Sigma D & \xrightarrow{\Sigma c} & \Sigma M\overline{D} \\
\downarrow{\scriptstyle \psi_0} & & \downarrow{\scriptstyle \beta} \quad (*) & & \downarrow{\scriptstyle \overline{\beta}} \\
T0 & \xrightarrow{op} & D & \xrightarrow{c} & M\overline{D} \\
\downarrow{\scriptstyle \tilde{T}?} & & \downarrow{\scriptstyle s} & & \downarrow{\scriptstyle M\alpha^{-1}} \\
MBT0 & \xrightarrow{MBop} & D & \xrightarrow{(\overline{B}c)^\dagger} & MB\overline{D}
\end{array} \tag{2}$$

The Σ-algebra structure $\overline{\beta}$ is induced in a similar manner to β, via the final $\overline{B}$-coalgebra morphism $\overline{\beta}_{TM\overline{D}}$ from $TM\overline{D}$ into $\overline{D}$: $\overline{\beta} = \overline{\beta}_{TM\overline{D}} \circ \psi_{M\overline{D}} \circ \Sigma\eta^T$. This gives an algebraic structure to the denotational model, and thus by initiality of $T0$, induces a denotational map – a Σ-algebra morphism – from $T0$ into $M\overline{D}$.

The operational map in our semantics – given by the final $\overline{B}$-coalgebra morphism from the operational model $T0$ into $\overline{D}$ (in $\mathsf{Kl}(M)$) – factors through the original one, op, as follows. The bottom-right square commutes by definition of c' as a $\overline{B}$-coalgebra morphism. By definition op is an MB-coalgebra morphism, hence Jop is also a $\overline{B}$-coalgebra morphism; thus the composition $c' \circ Jop$, equal to $c \circ op$ in the underlying category, is a $\overline{B}$-coalgebra morphism. By finality of $\overline{D}$ as a $\overline{B}$-coalgebra, $c \circ op$ is necessarily equal to the final $\overline{B}$-coalgebra morphism from $T0$ into $M\overline{D}$.

Now we aim to show the top-right square (*) commutes – i.e. that c is a Σ-algebra morphism. This would imply the denotational maps and operational maps coincide, giving adequacy and compositionality as for the original semantics. This is because op is known to be a Σ-algebra morphism, so the operational map $c \circ op$ would also be a Σ-algebra morphism, so must coincide with the denotational map.

The Σ's in condition (*) may be replaced with T's as follows, giving the square (+) below. (The top line is β, the bottom $\overline{\beta}$.)

$$\begin{array}{ccccccc}
\Sigma D & \xrightarrow{\Sigma\eta^T} & \Sigma TD & \xrightarrow{\psi_D} & TD & \xrightarrow{\beta_{TD}} & D \\
\downarrow{\scriptstyle \Sigma c} & & \downarrow{\scriptstyle \Sigma Tc} & & \downarrow{\scriptstyle Tc} \quad (+) & & \downarrow{\scriptstyle c} \\
\Sigma M\overline{D} & \xrightarrow{\Sigma\eta^T} & \Sigma TM\overline{D} & \xrightarrow{\psi_{M\overline{D}}} & TM\overline{D} & \xrightarrow{\overline{\beta}_{TM\overline{D}}} & M\overline{D}
\end{array}$$

The first square is the image under Σ of the naturality of η. The second commutes because ψ_X, the Σ-algebra structure of TX – the free Σ-algebra over X – is a natural transformation. This is an easy consequence of the definition of T by adjunction, $T = UF$.

One may interpret the condition (+) as an abstract constraint on the abstract OS ρ as follows. TD describes terms over the 'old' denotational model, whose arguments may introduce effects at each transition step. The upper path assigns an overall effect-tree to these terms, based on their behaviour given by ρ. The lower path (via Tc) collects the effects of each argument into its *first* execution step, giving an element of $TM\overline{D}$, and then similarly assigns an overall effect-tree. Thus, (+) implies that the stage at which effects occur (in the arguments of terms TD) is irrelevant, as they might as well all be in the first step.

4.4 A Condition on Cones

One may characterise the condition (+) somewhat more concretely, in terms of cones in the Kleisli category. For simplicity we assume the initial sequence of B in $\mathcal{C}$ converges after ω steps (as it does for $BX = V + A \times X$). Note that left-strictness implies $M0 = 1$, so 0 is the final object in $\mathsf{Kl}(M)$ ([3, Lemma 2.3.5]).

Definition 4.2 The *cone generated by a $\overline{B}$-coalgebra* $\langle X, \gamma\rangle$ (over the final sequence up to ω) consists of the arrows $(\gamma_n : X \to \overline{B}^n 0)_{n<\omega}$ obtainable by composition in the following diagram:

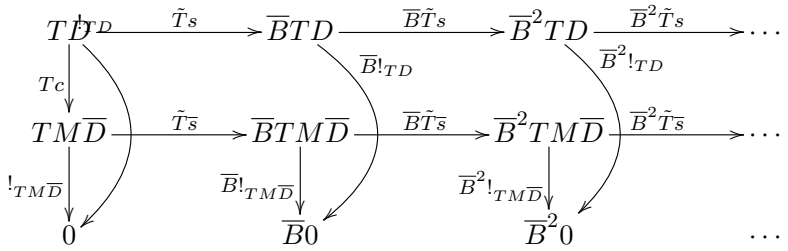
$$\begin{array}{ccccccc}
X & \xrightarrow{\gamma} & \overline{B}X & \xrightarrow{\overline{B}\gamma} & \overline{B}^2X & \xrightarrow{\overline{B}^2\gamma} & \cdots \\
\downarrow{\scriptstyle !_X} & & \downarrow{\scriptstyle \overline{B}!_X} & & \downarrow{\scriptstyle \overline{B}^2!_X} & & \\
0 & \xleftarrow{!_{\overline{B}0}} & \overline{B}0 & \xleftarrow{\overline{B}!_{\overline{B}0}} & \overline{B}^2 0 & \xleftarrow{\overline{B}^2!_{\overline{B}0}} & \cdots
\end{array}$$

The limit-colimit coincidence described in [4] implies that $(\overline{s}_n : \overline{D} \to \overline{B}^n 0)_{n<\omega}$ is a limiting cone [3]. It is straightforward to show that the $\overline{B}$-coalgebra morphism from any $\langle X, \gamma\rangle$ into $\overline{D}$ must coincide with the mediating morphism between their generated cones.

Revisiting the condition (+), the path $c \circ \beta_{TD}$ is a composition of $\overline{B}$-coalgebra morphisms, so it is also one, and must coincide with the mediating morphism from $((\tilde{T}s)_n)$ to $(\overline{s}_n)$. In the other path, $\beta_{TM\overline{D}}$ coincides with the mediating morphism from $((\tilde{T}\overline{s})_n)$ to $(\overline{s}_n)$, so precomposing with Tc gives another cone over the final sequence $((Tc \circ \tilde{T}\overline{s})_n)$, with mediating morphism $\beta_{TM\overline{D}} \circ Tc$.

Our strategy is to show this cone is the same as the cone generated by $\langle TD, \tilde{T}s\rangle$ – i.e., that $(\tilde{T}s)_n = (Tc \circ \tilde{T}\overline{s})_n$. This would imply the mediating morphisms are

the same, giving equality of the upper and lower paths. The situation is depicted below; we must show the two paths $TD \to \overline{B}^n 0$ coincide for all n.



We will focus on the case where ρ is in evaluation-in-context format, a perspective which is often helpful for specifying effectful operational semantics – (cf. [8,14]):

Definition 4.3 An abstract OS ρ is *in evaluation-in-context format* if it arises as an extension of effectless operational rules corresponding to the following templates:

$$\frac{x_1 \to x_1'}{\sigma(x_1,\ldots,x_n) \to \sigma(x_1',\ldots,x_n)} \qquad \frac{x_1 \xrightarrow{\checkmark} \underline{v}}{\sigma(x_1,\ldots,x_n) \to t} \text{ or } \frac{x_1 \xrightarrow{\checkmark} \underline{v}}{\sigma(x_1,\ldots,x_n) \to \underline{u}}$$

$$\frac{}{\sigma(x_1',\ldots,x_n) \to t} \text{ or } \frac{}{\sigma(x_1',\ldots,x_n) \to \underline{v}}$$

where t is an arbitrary term over $\underline{v}$ and the x_i.

Thus a term's behaviour depends on at most one subterm – without loss of generality, the first. It is executed in its place until termination, at which point the term evolves to another term depending on the final value.

Theorem 4.4 *($\mathcal{C} = \mathsf{Set}$) For $BX = V + X$, $MX = G_{\tilde{\Delta}}X$, and ρ in evaluation-in-context format, we have $(\tilde{T}s)_n = (Tc \circ \tilde{T}\overline{s})_n$. Thus condition (+) holds, and the denotational and operational semantics induced by the initial and final horizontal (co)algebra morphisms in (2) coincide.*

Proof. First, note that the horizontal arrows $TX \to \overline{B}^n TX$ for $X = D, M\overline{D}$ have (underlying) codomain MB^nTX, i.e. effect-trees over n-step behaviour traces; their leaves may be terminal traces – ending with a value in V – or not, ending with a term in TX. When the horizontal arrows are composed with vertical $\overline{B}^n X \to \overline{B}^n 0$, the non-terminal leaves are replaced with $\bot$. (This may be proven by induction on n, using the fact that the distributive law for $G_{\tilde{\Delta}}$ and B maps non-terminal $\mathsf{inr}(M0) \in BM0$ into $\bot$.)

Thus, for any n, to show the two paths $TD \to \overline{B}^n 0$ agree, it is enough to show that when applied to any term $t(d_1,\ldots) \in TD$, the horizontal paths to $\overline{B}^n TD$ and $\overline{B}^n TM\overline{D}$ produce effect-trees which share the same terminal leaves (and agree in shape before those leaves). This is because all non-terminal leaves will be mapped to $\bot$ by the ensuing vertical arrows (and any resulting all-$\bot$ subtrees removed by left-strictness as given by the definition of $G_{\tilde{\Delta}}$).

We prove this by induction on n. The $n = 0$ case is immediate as both vertical arrows map into the final object 0 in $\mathsf{Kl}(M)$; now assume the inductive hypothesis for $n-1$. The action of both horizontal paths consist of iterating $\overline{B}$-coalgebra maps induced by the derivation rules of ρ, applied to syntax terms over D or $M\overline{D}$. By assumption, ρ is obtained from effectless rules, which are single-premise and the premise must be the first argument.

When applying ρ to a syntax term $\sigma(x_1, \ldots)$, there are several cases to consider. If σ is an effectless command, and the behaviour of x_1 in MBX is relevant ($dep(\sigma) = 1$), the effectless rules are applied to each leaf of x_1's behaviour (in BX); the rule conclusions become the leaves of the behaviour of $\sigma(x_1, \ldots)$. As the rule conclusions are effectless, so are any further deductions.

If there is no dependency on x_1 ($dep(\sigma) = 0$), the behaviour of $\sigma(x_1, \ldots)$ is given directly by the rule conclusion. If σ is an effect, a layer of branching will be introduced, and $\sigma(x_1, \ldots)$ will evolve to some x_i on each branch.

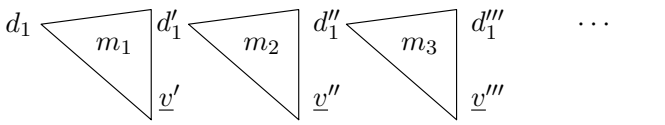
As a result, we may class any effectless derivation – as occurs at the leaves of the behaviour of a general term $t(x_1, \ldots)$ – into the following four types. (The conventions for $'$ will become clearer later.)

$$\frac{x_1 \to x_1'}{\vdots}{t(x_1, \ldots) \to t(x_1', \ldots)} \qquad \frac{\overline{\sigma(x_1, \ldots) \to t_0(y_1, \ldots)}}{\vdots}{t(x_1, \ldots) \to t'(y_1', \ldots) \text{ or } \underline{u}}$$

$$\frac{x_1 \to \underline{v}}{\vdots}{t(x_1, \ldots) \to \underline{u}} \qquad \frac{x_1 \to \underline{v}}{\vdots}{t(x_1, \ldots) \to t'(y_1', \ldots)}$$

By assumption on ρ, the top-left is the only form of derivation possible from a premise $x_1 \to x_1'$; all other arguments of t are fixed throughout. If the premise is $x_1 \to \underline{v}$ for some $\underline{v}$, the rule conclusions may similarly indicate termination (bottom-left) or give rise to a new term with a maybe-new first argument, $t'(y_1', \ldots)$ (bottom-right). Similarly, effect syntax and premiseless terms give rise to terms $t'(y_1', \ldots)$, but the former extends the effect-tree.

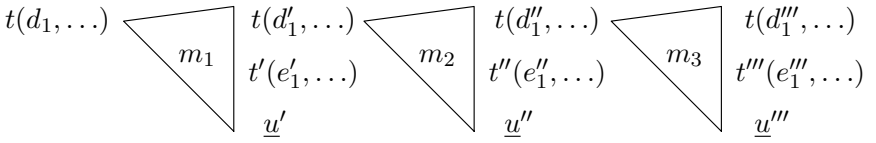
Such deductions define the coalgebra structure-maps of TD and $TM\overline{D}$; we consider how these maps are iterated in terms of these deductions. Take a term $t(d_1, \ldots)$ in TD, and consider the action of the horizontal map $TD \to \overline{B}^n TD$. We may represent d_1 schematically as follows:



Here, m_1 represents the effectful behaviour of $d_1 \in D$. Its leaves, elements of BD, are either elements of D – collectively represented by the label d_1' – or terminal values represented by $\underline{v}$. Likewise, m_2 represents the effectful behaviour of each leaf d_1', and so on. (Each d_1' has its own effect-tree m_2, but to keep diagrams simple, we

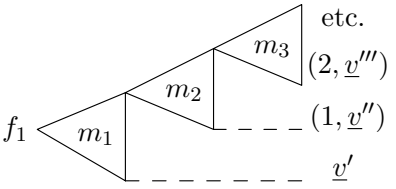
do not attempt to represent this information.)

We first focus on the case where the behaviour of the term $t(d_1, \ldots)$ depends on its first argument. Its effectful behaviour resembles that of d_1 – 'the same branches' – but with each leaf l derived via the operational rules, with premises given by the corresponding leaf r of d_1. Depending on whether they are terminal values $\underline{v}$ or other elements of D, l will take one of the forms listed above (with x, y relabelled d, e). Thus we may partially represent the behaviour of $t(d_1, \ldots)$ in terms of that of d_1 as follows, omitting continuations which depend on other arguments e', e'', etc:



(3)

Note that the terms t' and denotations e' depend only on t and the corresponding leaf $\underline{v}'$ of d_1, and similarly for t'', e'' and so on. Now let us consider the other 'horizontal' path $TD \xrightarrow{Tc} TM\overline{D} \to \overline{B}^n TM\overline{D}$. We write f_1 for the result in $M\overline{D}$ of applying c to (d_1), and similarly $g_1 = c(e_1)$, etc. It may be represented thus:



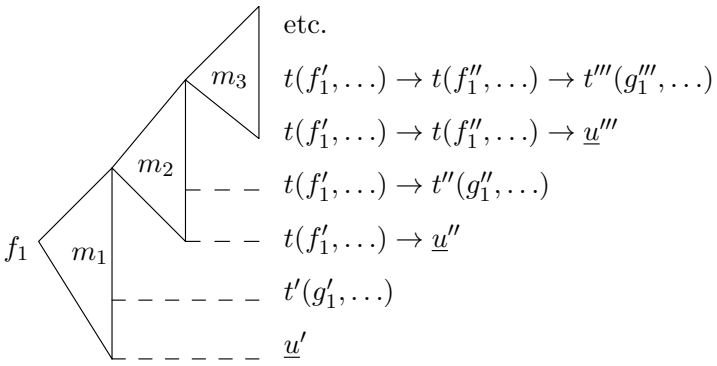
where m_i represent the effect-trees occurring in the behaviours of the i^{th} successors of d_1; they have been combined into a single layer of effects. We write $(m, \underline{v})$ for the element of the initial algebra $\overline{D}$ which terminates in m steps with value $\underline{v}$.

Now let us consider the same term $t(f_1, \ldots)$ as before, but with arguments d_i replaced by their images under c. As before, the behaviour of $t(f_1, \ldots)$ is obtained by applying the effectless rules to each leaf of f_1, retaining the conclusions in an effect-tree of the same shape.

Writing $^{(m)}$ for m primes$'^{\cdots\prime}$, first note that for every m, and every leaf $\underline{v}^{(m)}$ of the m^{th} successor of $d_1^{(m)}$ of d_1, there is a corresponding leaf $(m-1, \underline{v})$ of f_1 (where we identify $(0, \underline{v})$ with $\underline{v}$). Now for any series of effectful transitions $t(d_1, \ldots) \to \ldots \to t(d_1^{(m-1)}, \ldots) \to \underline{u}^{(m)}$ or $t^{(m)}(e_1^{(m)}, \ldots)$, the premises of their derivations must respectively be $d_1 \to d_1'$, $d_1' \to d_1''$, …, $d_1^{(m-1)} \xrightarrow{\checkmark} \underline{v}^{(m)}$ (for some successor d_1' of d_1, and so on). In addition, by naturality of ρ the arguments $e_i^{(m)}$ of $t^{(m)}$ must come from those of $t(d_1, \ldots)$. By the previous remark, there is a corresponding leaf $(m-1, \underline{v}^{(m)})$ of f_1 with the same transitions: $(m-1, \underline{v}^{(m)}) \to \ldots \xrightarrow{\checkmark} \underline{v}^{(m)}$. This allows us to derive the following transitions:

$$t((m-1, \underline{u}^{(m)})) \to \ldots \to t(1, \underline{u}^{(m)}) \to \quad \underline{u}^{(m)} \text{ or } t^{(m)}(g_1^{(m)}, \ldots)$$

This implies the behaviour of $t(f_1, \ldots)$ is as follows, where the $\underline{u}^{(m)}$ match the corresponding leaves of (3), $t^{(m)}$ match the corresponding syntax-terms, and $f^{(m)}, g^{(m)}$ are the image under c of $d^{(m)}, e^{(m)}$.



Now we show the required induction step. First, consider the case where the behaviour of $t(d_1, \ldots)$ depends on d_1. The diagram (3) shows that after iterating the behaviour of $t(d_1, \ldots)$ for n steps, the resulting terminal leaves can come from two sources – either from the terminal leaves $\underline{v}^{(m)}$ of d_1 (for $m < n$), resulting in terminal leaves $\underline{u}^{(m)}$ in the behaviour of $t(d_1, \ldots)$; or from the unshown execution of successor terms, $t^{(m)}(e_1^{(m)}, \ldots)$.

The above diagram shows that for every terminal leaf arising from the former case, there is a matching leaf in the behaviour of $t(f_1, \ldots)$, and vice versa. For the latter case, it also shows that the successor terms $t^{(m)}(e_1^{(m)}, \ldots)$ reachable in m steps by $t(d_1, \ldots)$ correspond with terms $t^{(m)}(g_1^{(m)}, \ldots)$ reachable by $t(f_1, \ldots)$ in m steps. Any terminal leaves arising in n steps via these successors must arise in less than n steps when evaluating them directly; so the inductive hypothesis implies both horizontal paths agree at those leaves too.

Finally, if the behaviour of term $t(d_1, \ldots)$ does not depend on d_1, the derivation of its behaviour $t(d_1, \ldots) \to t'(e_1', \ldots)$ may be repeated exactly for the term obtained by applying Tc: $t(f_1, \ldots) \to t'(g_1', \ldots)$, introducing the same effects. The inductive hypothesis applied to these successor terms tells us that for $t(d_1, \ldots)$ and $t(f_1, \ldots)$, the terminal leaves produced by n steps of behaviour agree.

□

Example 4.5 By inspection of the operational rules for stateless while, we see the abstract OS for the full while language are in evaluation-in-context format; Theorem 4.4 implies compositionality of the effectful operational semantics illustrated by Examples 3.10, 3.12 and 3.14. Adequacy follows from the fact that two programs are operationally equivalence $p \cong_{op} q$ iff they are identified by the final coalgebra map, which coincides with the denotational map; hence denotational equivalence implies operational equivalence.

However, the induced denotational semantics is more fine-grained than the traditional semantics for `While`, in two respects. Firstly, we distinguish expressions like 0 and $0 + 0 + 0$ because they respectively terminate in 0 and 2 steps, even though

the final value is the same; this is usually not the desired semantics.

Secondly, in the absence of equations relating effect-trees, denotations such as $\mathrm{wr}_{x,2}(1,*)$ and $\mathrm{wr}_{x,2}(\mathrm{wr}_{x,2}(1,*))$ are inappropriately distinguished. However, the semantics does abstract away from the timing of effects provided they stay in the right order, as programs p_1 and p_2 of Example 4.1 demonstrate.

5 Discussion and Future Work

In summary, we started with premise-supported operational specifications for multi-sorted, effectless languages, and demonstrated how to extend them with purely syntactic effects. This gives an operational model as a coalgebra in a Kleisli category, if the behaviour functor has a lifting. Under certain conditions, the final Kleisli-coalgebra exists, and we define an operational semantics by the unique Kleisli-coalgebra morphism into the final Kleisli-coalgebra. We may take this coalgebra to be the denotational model, by giving it an algebra structure.

By mapping from the final coalgebra in the underlying category into the final Kleisli-coalgebra, we showed that our semantic maps are as a quotient of those in Turi and Plotkin's framework, and that our denotational semantics is adequate when the effectful operational semantics is in the evaluation-in-context format.

However there are several key omissions that we would like to address. Firstly, we have not considered equations on effects. Equations for finite effect-trees $T_{\tilde{\Delta}}X$ amounts to quotienting the algebra; in a Lawvere theory, they correspond to sketches [1]. A less syntax-driven approach would be to represent effect-trees via the free model of the Lawvere theory in the base category. This would mean discarding the 'final Δ-coalgebra' approach of Definition 2.14 and taking M as the free Δ-algebra functor in Cppo; with equations, this corresponds to a free model functor on Cppo. This approach may also clarify how one might guarantee left-strictness of the resulting Kleisli category in a more canonical way.

Secondly, a more conceptual, abstract adequacy proof would allow easier generalisation. It may help to have a more semantic characterisation of evaluation-in-context, possibly in terms of the operational specifications ρ rather than on the induced operational models $TD, TM\overline{D}$.

Also, it is not yet clear how useful dependency-functions will be for obtaining effectful operational semantics ρ^{Eff} from effectless ones. They work well for single-premise languages like while, and the multi-premise situation seems more straightforward when the effects are commutative. Otherwise, it may be easier to specify them directly; more examples will clarify this situation.

Lastly, our denotational semantics is sensitive to the number of steps-to-termination, which may be too fine-grained for some applications. However, in a Kleisli category it may be possible to to discard this information.

Another key question we did not have space to discuss is the relationship of the above semantics with comodels [16], which implement effects and are suitable for e.g. global state and interactive I/O [20]. However, they do not account for non-determinism, unless we exclude equations. By contrast, the conventional form

of operational semantics for nondeterminism is obtained by imposing equations on the effect-trees in the operational model we obtain in this paper. For each comodel with carrier C there is a notion of 'comodel-induced operational model', this time a coalgebra in the Kleisli category of the side-effect monad with bottom, $S_C X = (\bot + C \times X)^C$. Analogously to $\mathsf{Kl}(G_{\tilde{\Delta}})$, one may give a Cppo-enrichment – so that there is a final Kleisli-coalgebra which we might take as a denotational model, and the target of an operational equivalence.

Given the effectful operational model – a $G_{\tilde{\Delta}}B$-coalgebra – one may obtain a comodel-induced operational model, an $S_C B$-coalgebra, via a natural transformation $\epsilon : G_{\tilde{\Delta}} \to S_C$ allowing comodels to 'traverse' trees and produce a result x in X, as well as a new comodel state.

Example 5.1 The comodels for global state are transition systems with implementations of variable lookup and update; the cofree comodel is the canonical example, the standard implementation S of global store with carrier $\mathbb{N}^L$. Given a comodel, applying ϵ to the operational model for `While` in Example 3.12, we obtain an arrow which, transposed, is of form $T0 \times C \to BT0 \times C$. This closely corresponds to the standard transition-form of `While` programs: $\langle p, c\rangle \to \langle p', c'\rangle$ or $\langle p, c\rangle \xrightarrow{\surd} c'$ where each c in C is a state of the comodel – the 'store'.

Consider $p = \texttt{if}\ (x = 1)\ \texttt{then}\ \{y{:=}2\}\ \texttt{else}\ \{\texttt{skip}\}$ as in example 3.12, with the same shorthand notation. Suppose the store consists of two variables and is initially $c = [x : 1, y : 0]$:

$$\begin{array}{lll} & \langle [x:1,y:0], & \texttt{if}\ (x=1)\ \texttt{then}\ \{y{:=}2\}\ \texttt{else}\ \{\texttt{skip}\}\rangle \\ \to & \langle [x:1,y:0], & \texttt{if}\ (1=1)\ \texttt{then}\ \{y{:=}2\}\ \texttt{else}\ \{\texttt{skip}\}\rangle \\ \to & \langle [x:1,y:0], & \texttt{if}\ (=_1\ (1))\ \texttt{then}\ \{y{:=}2\}\ \texttt{else}\ \{\texttt{skip}\}\rangle \\ \to & \langle [x:1,y:0], & y{:=}2\rangle \\ \to & \langle [x:1,y:2], & *\rangle \end{array}$$

The map ! into the final S_C-coalgebra is of type $T0 \to (\bot + C \times D)^C$, where $D = (\mathbb{N} \times \mathbb{N}, \mathbb{N} \times \mathbb{B}, \mathbb{N} \times 1)$ as explained after Example 3.12. By uniqueness, one may check that it gives for each program p and initial comodel state c, either (a.) a tuple (c', n, v) giving the final comodel state, steps to termination, and output value; or (b.) $\bot$ if p diverges in state c. Indeed, one finds

$$!(p)([x:1,y:0]) = ([x:1,y:2], 4)$$

in agreement with the above behaviour.

It may be that adequacy of the effectful semantics in this paper implies adequacy of the comodel-based semantics; if the combination of effect-trees comb is by monad strength, it may be possible to exploit the fact that ρ^{Eff} is defined 'in the same way for each monad' in terms of monad morphisms.

References

[1] M. Barr and C. Wells. Toposes, triples, and theories. *Reprints in Theory and Applications of Categories no.1*, pages 1–289, 2005.

[2] J. A. Goguen, J. W. Thatcher, E. G. Wagner, and J. B. Wright. Initial algebra semantics and continuous algebras. *J. ACM*, 24:68–95, January 1977.

[3] I. Hasuo. *Tracing Anonymity with Coalgebras*. PhD thesis, Radboud University, Nijmegen, March 2008.

[4] I. Hasuo, B. Jacobs, and A. Sokolova. Generic trace semantics via coinduction. *Logical Methods in Computer Science*, 3(4), 2007.

[5] M. Hyland, G. Plotkin, and J. Power. Combining effects: sum and tensor. *Theor. Comput. Sci.*, 357:70–99, 2006.

[6] M. Hyland and J. Power. Discrete lawvere theories and computational effects. *Theor. Comput. Sci.*, 366(1-2):144–162, 2006.

[7] B. Jacobs. Trace semantics for coalgebras. *Electr. Notes Theor. Comput. Sci.*, 106:167–184, 2004.

[8] P. Johann, A. Simpson, and J. Voigtländer. A generic operational metatheory for algebraic effects. In *Proc. LICS 2010*, pages 209–218. IEEE Computer Society, 2010.

[9] B. Klin. Bialgebraic methods and modal logic in structural operational semantics. *Inf. Comput.*, 207(2):237–257, 2009.

[10] B. Klin and V. Sassone. Structural operational semantics for stochastic process calculi. In R. M. Amadio, editor, *Proc. FoSSaCS 2008*, volume 4962 of *Lecture Notes in Computer Science*, pages 428–442. Springer, 2008.

[11] R. Matthes and T. Uustalu. Substitution in non-wellfounded syntax with variable binding. *Theor. Comput. Sci.*, 327(1-2):155–174, 2004.

[12] E. Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1):55–92, 1991.

[13] H. R. Nielson and F. Nielson. *Semantics with Applications: A Formal Introduction*. Wiley Professional Computing. Wiley, 1992.

[14] G. D. Plotkin and J. Power. Adequacy for algebraic effects. In *FoSSaCS '01: Proceedings of the 4th International Conference on Foundations of Software Science and Computation Structures*, pages 1–24, London, UK, 2001. Springer-Verlag.

[15] G. D. Plotkin and J. Power. Notions of computation determine monads. In *FoSSaCS '02: Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structures*, London, UK, 2002. Springer-Verlag.

[16] G. D. Plotkin and J. Power. Tensors of comodels and models for operational semantics. *Electr. Notes Theor. Comput. Sci.*, 218:295–311, 2008.

[17] M. B. Smyth and G. D. Plotkin. The Category-Theoretic Solution of Recursive Domain Equations In *SIAM J. Comput.*, 11(4):761–83, 1982.

[18] J. Power. Enriched lawvere theories. *Theory and Applications of Categories*, 6(7):83–93, 1999.

[19] J. Power. Countable lawvere theories and computational effects. *Electr. Notes Theor. Comput. Sci.*, 161:59–71, 2006.

[20] J. Power and O. Shkaravska. From comodels to coalgebras: State and arrays. *Electron. Notes Theor. Comput. Sci.*, 106, 2004.

[21] J. Power and D. Turi. A coalgebraic foundation for linear time semantics. In *In Category Theory and Computer Science*. Elsevier, 1999.

[22] J. J. M. M. Rutten and D. Turi. Initial algebra and final coalgebra semantics for concurrency. In J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors, *Proc. REX School/Symposium 1993*, volume 803 of *Lecture Notes in Computer Science*, pages 530–582. Springer, 1994.

[23] D. Turi. *Functorial Operational Semantics and its Denotational Dual*. PhD thesis, Free University, Amsterdam, June 1996.

[24] D. Turi and G. D. Plotkin. Towards a mathematical operational semantics. In *LICS*, pages 280–291, 1997.